METHOD FOR ENABLING A PROGRAM WRITTEN IN UNTRUSTED CODE TO INTERACT WITH A SECURITY SUBSYSTEM OF A HOSTING OPERATING SYSTEM

Technical Field

The present invention relates generally to enabling a program written in untrusted code (e.g., Java) to access a resource managed by a closed operating system (e.g., Windows NT).

Description of the Related Art

Java, originally developed by Sun Microsystems, is an object-oriented, multi-threaded, portable, platform-independent, secure programming environment used to develop, test and maintain software programs. Java programs have found extensive use on the World Wide Web, which is the Internet's multimedia information retrieval system. These programs include full-featured interactive, standalone applications, as well as smaller programs, known as applets, that run in a Java-enabled Web browser or applet viewer.

Initially, programs written in Java found widespread use in Internet applications. As a result of this browser-centric focus, security concerns raised by Java primarily involved the security of the Java sandbox and the origin of the executable code (namely, the class). More recently, Java is beginning to move out of the browser and into server backend environments. With this change, it becomes necessary to consider security

concerns associated with more traditional environments, e.g., identifying the user of the Java program and what privileges should be granted to that user. To this end, it has been proposed to define a Java Authentication Service framework as a standard extension on top of the Java Development Kit (JDK) 1.2.

Java's early acceptance was driven largely by the Web and desire for active content on Web servers, but its continuing incorporation into information technology infrastructures has been somewhat limited by Java's lack of integration with underlying operating system services. Meanwhile, on a parallel track, the Windows NT operating system, with its support from fairly sophisticated security mechanisms (for a commodity operating system) has been increasingly used as the base for new applications.

Given the nature of the NT security mechanisms, it has not been possible to allow Java programs to access NT operating system resources. Because of the "closed" nature of Windows NT, a user of a client machine may only log on against an account held at the machine, at a server running the Windows NT operating system, or at any other servers that are "trusted" by the NT server that the client is configured against. Only these options are supplied to the user during the logon process, and there are no practical interfaces to allow user authentication from non-native server domains. This closed

architecture, together with the Java security paradigm, makes it difficult to interface a Java program to a Windows NT resource.

In particular, Windows NT does not allow normal programs to run under an identity other than the one in which they started. Once a user logs in, all programs inherit that original identity. Specifically, Windows NT enforces this prohibition by requiring that callers of the LogonUser API, which results in a new access token, must be running with a given privilege, and this privilege is only available to the most trusted of users.

It would be desirable to provide a bridge between ease of programming with Java and the rich security model afforded by NT.

The present invention solves this problem.

BRIEF SUMMARY OF THE INVENTION

It is an object of the present invention to allow a Java program to access NT operating system resources under the identity of the user running the Java program.

A more specific object of this invention is to facilitate Windows NT login from an Java-based authentication service.

Still another object of the invention is to allow application servers running Java programs to run each program as a separate thread and have each thread run as a different NT user.

Another more specific object of the invention is to provide a mechanism that binds a particular Windows NT identity to a particular thread executing in a Java Virtual Machine (JVM).

Another object of the invention is to enable Java programs to take advantage of the rich security model available from the NT operating system platform.

A more general object of the present invention is to enable a program written in untrusted code to login to and access a resource within a closed operating system environment.

Yet another general object of this invention is to provide a mechanism that enables an enterprise to

leverage its investments both in Java and in NT security protections.

According to the invention, a program written in untrusted code (i.e. code that is not part of a trusted computing base) is enabled to access a native operating system resource through a staged login protocol. In operation, a trusted login service listens, e.g., on a named pipe, for requests for login credentials. In response to a login request, the trusted login service requests a native operating system identifier. The native operating system identifier is then sent to the program. Using this identifier, a credential object is then created within an authentication framework. The credential object is then used to login to the native operating system to thereby enable the program to access the resource.

The technique enables the program written in untrusted code (e.g., Java) to access the operating system resource (e.g., supported in Windows NT) under the identity of the user running the program.

The foregoing has outlined some of the more pertinent objects and features of the present invention. These objects should be construed to be merely illustrative of some of the more prominent features and applications of the invention. Many other beneficial

results can be attained by applying the disclosed invention in a different manner or modifying the invention as will be described. Accordingly, other objects and a fuller understanding of the invention may be had by referring to the following Detailed Description of the Preferred Embodiment.

BRIEF DESCRIPTION OF THE DRAWINGS

For a more complete understanding of the present invention and the advantages thereof, reference should be made to the following Detailed Description taken in connection with the accompanying drawings in which:

Figure 1 is a block diagram of the main operating processes of the present invention;

Figure 2 is a detailed flowchart illustrating the operation of the inventive protocol;

Figure 3 is a flowchart illustrating the process steps of a service thread executing in the trusted code service routine;

Figure 4 is a flowchart illustrating the process steps of the commit routine of the Java authentication service; and

Figure 5 illustrates a conventional client-server operating environment in which the present invention is implemented.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENT

As described above, the present invention enables a Java program to access a Windows NT operating system resource under the identity of the person running the Java program. Although not meant to be limiting, the invention may be implemented in a Web application server running Java-based programs. As will be seen, the invention allows the server to run each Java program as a separate thread and to have each thread run as a different Windows NT user. To this end, it is assumed that each Java program (or each user thereof) must perform a Windows NT login from a Java-based authentication service. The present invention preferably is implemented underneath or "under the covers" as this Windows NT login takes place.

Figure 1 illustrates the operating environment in which the present invention may be implemented. As will be described, there are three (3) main functional components that are used to enable Windows NT login from a Java-based authentication service. In particular, there are two (2) components of untrusted code, namely, components referred to NTLoginModule **30** and MUJLogin.dll **32**, and one component of trusted code, i.e. MUJService.exe **34**. The NTLoginModule **30** may be written

in Java. The MUJLogin.dll component **32** may be a native code library. The program requesting access to the native operating system is identified by reference numeral **35**. In an illustrative embodiment, the program **35** is written in Java and the native operating system is Windows NT. With respect to the operating system kernel and its resources, the Java program **35** is untrusted code.

The NTLoginModule **30** preferably includes a set of application programming interfaces (APIs), namely, login() API **38**, commit() API **40**, abort() API **42**, and logout() API **44**. Collectively, these APIs comprise an authentication framework. In a preferred embodiment, the authentication framework is compliant with the Java Authentication and Authorization Services framework, which is a new standard extension on top of the Java 1.2 Java Development Kit (JDK). Alternatively, the authentication framework is any pluggable authentication mechanism (PAM).

The MUJLogin.dll (multi-user Java login module) **32** includes an initialization routine **46**, as well as a set of corresponding APIs, namely, the login() API **48**, commit() API **50**, abort() API **52**, and the logout() API **54**.

The MUJService.exe (multi-user Java login service) **34** includes a listener routine **56** and a set of one or more service threads **58**. The MUJService.exe **34** component encapsulates the LogonUser API **60**, which is an API that can only be called by trusted code.

The inventive protocol is now described in the flowchart of **Figure 2**. The code involved in these functions is untrusted. It is assumed that the initialization routine **46** of MUJLogin dll **32** is initialized to open a service pipe and to create a uniquely-named response pipe. Also, the MUJService.exe component **34** is initialized to create a service pipe with a particular name that can be discovered. The routine then begins at step **70** with a call to the login() API **38** of the NTLoginModule **30**. At step **72**, the API prompts for the user to enter his or her ID and password. When that information is entered, the login() API **38** calls the native code login() API **48**, passing the user-entered information. This is step **74**, which transfers control to the native code component. The login() API **48** of MUJLogin.dll **32** then continues at step **76** to format a request and to send the request to MUJService.exe **34** through the service pipe initialized by the

initialization routine **46**. Control then continues at the MUJService.exe component **34**.

In particular, the MUJService.exe component **34**, which had been listening on its service pipe, recognizes that a request has been received. This is step **78**. At step **80**, the MUJService.exe creates a service thread to process the request. This routine then loops back to listen for further requests. A test is then performed at step **82** to determine whether the thread is ended. If not, the routine cycles. If so, at step **84**, the answer to the request sent by the login() API **48** is sent back to this API. At this point, control returns back to the MUJLogin.dll component **32**.

Figure 3 is a flowchart of the service thread. The code involved in these functions is trusted. The routine begins at step **81** to receive certain data, namely, a verb (logon), a userid (new_user), a password, and a reply pipename. At step **83**, the service thread invokes the LogonUser() API. If the invocation is successful, the routine continues at step **85** to open the response pipe. At step **87**, the service thread invokes an ImpersonateLoggedOnUser() API. The user's new identity, new_user, or any other text is then written to the

response pipe at step **89**, which is then closed at step **91**. At step **93**, the RevertToSelf() API is invoked, which reverts the thread back to its previous identity. The service thread terminates and control then returns back to the MUJLogin.dll **32**.

Referring now back to **Figure 2**, the service was running as the new_user when it wrote to the response pipe. The MUJLogin.dll API then continues its operation. At step **85**, the ImpersonateNamedPipeClient() API is invoked. At step **86**, the login() API **48** invokes an OpenThreadToken() API. At step **88**, a DuplicateTokenEx() API is invoked to duplicate the token. The RevertToSelf() API is then invoked at step **90** to enable the thread to revert back to its original identity. At step **92**, the login() API **48** returns to the Java login() API **38** the duplicated token. Preferably, the duplicated token is an integer value and, in particular, an index into a process local table in the NT operating system. Control then returns back to the NTLoginModule **30**.

In particular, the login() API **38** in the NTLoginModule **30** then creates a Principal object at step **94**. At step **96**, the login() API **38** creates a Credential object. At step **98**, the integer value (namely, the

duplicated token) is stored in the Credential object. The login() API **38** then returns at step **100**. At this point, the login() API **38** has authenticated that the Java program can become an NT user and thus access resources in the native NT operating system environment.

The commit() API **40** of the NTLoginModule **30** is the functionality that is used to enable the Java program to become an NT user. **Figure 4** illustrates the functionality. When the commit() API **40** is invoked, the routine continues at step **102** to call the native commit() API. Control then passes again to the MUJLogin.dll component **32**. At step **104**, the native commit() API **50** is invoked. This API is then executed. At step **106**, the API locates the Credential. At step **108**, the API then retrieves the token. The routine then invokes an ImpersonateLoggedOnUser() API at step **110**, which returns control back to the NTLoginModule **30** to complete the processing.

Thus, according to a preferred embodiment, a Java program obtains access to a Windows NT operating system resource in a staged login process. A Windows NT service, which runs under a local system account, has the necessary authority to issue LogonUser calls. This service, however, can be accessed by normal programs

through either named pipes or remote procedure calls (RPCs). Accordingly, the present invention as explained above defines a protocol to pass the desired username and password from the Java program to the Windows NT service. In operation, the service listens on a well-known named pipe for "logon" requests. The service, upon receiving a call, then issues a LogonUser call to get credentials. To avoid the problem of cross-process transmission of an access token, the protocol passes the name of a uniquely-named named pipe on each logon request. The original caller (in this case, a dll) acts as a named-pipe server and listens for a response from the NT service on this pipe. Once the service has obtained the new access token, it issues an ImpersonateLoggedOnUser() call, which associates the new access token with the current service thread. The service has now effectively become that new user. The service then opens the named pipe whose name was transmitted to it and sends back a response (any data will do). The original Java program, which has been waiting on a response on its named pipe, then issues an ImpersonateNamedPipeClient() call, which allows any named pipe server to run under the authority of its caller to perform its actions. Because the NT service had changed to be the new user, the original Java program is now running as the new user.

Then, the original program (running as the new user), issues an OpenThreadToken() call on the current

thread, followed by a DuplicateTokenEx() call to duplicate the access token for the current thread. This operation creates a reference in the underlying kernel structures for the current process that allows the protocol to continue to reference this access token in the future. This token reference is saved, so that it can be handed back to the authentication framework for use as a credential. The current program then performs a RevertToSelf() call (which reverts to its previous identity), disconnects and closes the named pipe, and returns the token reference (an integer) back to the authentication framework. When the login chain finishes running, it calls back to the commit() API. The integer is then passed for use on the SetThreadToken() call. As a result, a change in the NT identity has been effected.

The inventive protocol thus allows the Java program to access the Windows NT operating system resource under the identity of the person running the Java program. This functionality enables Java programs to be successfully integrated with underlying NT operating system services. Thus, one illustrative operating environment of this invention is an application server (e.g., a Web server) running Java programs. This architecture is illustrated in **Figure 5**.

In this example, a plurality of client machines **10** access the application server **12** via a computer network

15 such as the Internet, an intranet, or some other computer network. A representative client machine is a personal computer that is x86-, PowerPC®- or RISC-based, that includes an operating system such as Windows NT, IBM® OS/2® or Microsoft Windows '95 or higher, and that includes a Web browser, such as Netscape Navigator 4.0 (or higher), having a Java Virtual Machine (JVM) and support for application plug-ins or helper applications. Typically, the server 12 is another personal computer or workstation platform that is Intel-, PowerPC®- or RISC®-based, and includes an operating system such as Windows® NT 4.0. The server runs Java programs 16a-16n to provide various services. Each Java program is capable of being executed in a separate thread. According to the present invention as previously described, each thread can run as a different NT user. This enables the operator of the server to leverage its investment in Java and in the underlying NT security protections.

The inventive protocol, however, is not limited to use on a Web server platform. Rather, the protocol may be implemented within an NT client or, more generally, within any operating environment in which the Java program seeks to obtain access to a native NT operating system resource. The inventive technique, however, is not limited to Java programs and Windows NT. The

technique may be practiced whenever it is desired to enable a program written in code that is not part of a trusted computing base to interact with a security subsystem of a hosting operating system. Further, the technique may be used with any programming architecture or language from which a callout into native code may be made. Thus, the program may be an ActiveX program, a program written in Visual Basic, or the like. Moreover, the given authentication framework utilized is not limited to that framework illustrated above. The authentication framework also may be any pluggable authentication mechanism known in the art (e.g., DCE PAM).

The present invention provides many advantages over the prior art. As noted above, it enables a program written in Java to interact with the security subsystem of a hosting operating system, namely, Windows NT, that normally does not allow programs to run under an identity other than the one in which they started. The invention may be implemented without making changes to the base Java Virtual Machine (JVM) on which the Java programs execute, and the protocol allows a multi-user framework inside of JVM on a very popular commodity operating system.

As has now been described, this invention provides a bridge between the ease of programming with Java and the rich security model available from NT. In particular, by

allowing Java programs to access operating system resources under the identity of the person running the Java program, the technique allows each of a set of Java programs running on an NT platform to execute in its own thread as a different NT user. As a result, the invention leverages both the investment that corporations have made in Java and the investments they have made in setting up proper security protections in NT.

One of the preferred implementations of the various routines described above is as a set of instructions (program code) in a code module resident in the random access memory of the computer. Until required by the computer, the set of instructions may be stored in another computer memory, for example, in a hard disk drive, or in a removable memory such as an optical disk (for eventual use in a CD ROM) or floppy disk (for eventual use in a floppy disk drive), or downloaded via a computer network.

In addition, although the various methods described are conveniently implemented in a general purpose computer selectively activated or reconfigured by software, one of ordinary skill in the art would also recognize that such methods may be carried out in hardware, in firmware, or in more specialized apparatus constructed to perform the required method steps.

Further, although the invention has been described in terms of a preferred embodiment in a specific

application environment, those skilled in the art will recognize that the invention can be practiced, with modification, in other and different hardware and operating system architectures with the spirit and scope of the appended claims. Thus, for example, while the present invention is preferably implemented to allow Java programs to access Windows NT resources, the principles of the invention are equally applicable with other known architectures. Once such example is a Java servlet environment.

Having thus described our invention, what we claim as new and desire to secure by Letters Patent is set forth in the following claims.